CS230

Gibberfish: Modeling Language by Detecting Nonsense

Bo Peng
Department of CEE
Stanford University
bpeng@stanford.edu

Matthew G. Mistele
Department of Computer Science
Stanford University
mmistele@cs.stanford.edu

Abstract

This paper asks: what if models of language were trained not by having them try to predict the next word, but predict whether the phrase or sentence as a whole was syntactically correct and made sense? How well would it perform, and what language structures would it learn along the way?

To that end, we built RNNs to predict whether a sequence of words is a well-formed English sentence. Our 2-layer word LSTM is 96% accurate at distinguishing valid sentences from sequences of words randomly sampled from the corpus. We analyzed its activations in search of learned structure representations and have preliminary visualization results.

1 Introduction

If done right, we conjectured, the activation of the hidden state neurons in an RNN trained simply to distinguish well-formed inputs from nonsense could capture different aspects of how human written sentence are structured, without any explicit part of speech tagging or other labeling more granular than a 0 or a 1 for the sequence as a whole. This motivated our creation of "gibberfish": RNNs that learn to model language by "fishing for gibberish".

The input to our algorithm is a string of up to 30 English words. We process the string into a sequence of lowercased inputs and use a LSTM recurrent neural network to output a prediction for whether the sentence was real (1) or randomly generated (0).

2 Related work

What makes a sentence "sensical" or "nonsensical" is a debated topic in linguistics. From the linguistic point of view, a nonsensical sentence refers to a sequence whose meaning cannot be interpreted by reader. Fowler [1] believes that a "nonsense" string is very hard for a reader to do “retrieval of the syntax of an utterance”. To automatically identify nonsensical sentences, recurrent neural networks have been proven a great tool, particularly through modeling the relationship between words in a sentence [2]. Zhang, etc. [3] developed an RNN model to capture the semantic relationships between different words in a sentence, such as cause-effect, component-whole, etc. Based on Zhang’s model, Wu [4] further improves the model, especially in over-fitting aspects, and applied the model to detect the nonsense part of a sentence. In both Zhang and Wu’s model, words are represented by a word embeddings [5], and an LSTM (long-short term memory) [6] model is applied. We used their work as a starting point for choosing features and models.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

3 Dataset and Features

Our dataset consists of 100,927 strings, half positive examples in the form of short English phrases and sentences from Tatoeba.org [8] and half negative examples ("invalid/nonsensical sentences") produced by sampling sequences of words from the corpus at random according to their occurrence frequency. All examples were lowercased to force the model to learn structure more interesting than "these capitalized words shouldn't appear in the middle of a sentence."

We had 99,080 (98.2%) examples in the train set, 184 (0.2%) in the dev set [1] , and 1663 (1.6%) in the test set, each evenly split between positive and negative examples.

For our word models, we removed punctuation from each example, tokenized the strings by whitespace, and replaced each word with a 50-dimensional GloVe pre-trained word embedding to create the input sequence. See 1 for the cost function the GloVe embeddings were pre-trained to minimize [5].

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \bar{w}_j + b_i + \bar{b_j} - \log X_{ij})^2 \tag{1}$$

(X_{ij} denotes the number of times word j occurs in the context of word i, w_j is word vector for j and $\bar{w_i}$ is the context word vector for i, b_i, b_j are the biases, and $f(X_{ij})$ is a weighting function.)

The input sequence for our character models consisted of one-hot encodings of the characters in the example.

Here are some examples from the dataset:

Input	Label
the committee has not yet arrived at a decision	1
a policeman ran past	1
this necktie does not match my coat	1
ugly really a a bee as	0
the her and california ugly american if away failures then	0
and it day take something unless long of extreme	0

4 Methods

4.1 Algorithms and Models

Our model sought to minimize the binary cross-entropy cost function 2,

$$J = -\frac{1}{M} \sum_{j=1}^{M} y^{(j)} \log \hat{y}^{(j)} + (1 - y^{(j)}) \log(1 - \hat{y}^{(j)}) \tag{2}$$

where $y^{(j)}$ is the ground truth label of the jth example and $\hat{y}^{(j)}$ is its predicted label.

To train model weights to minimize the loss, we used an Adam optimizer (short for "Adaptive Moment Estimation") [7], an extension of gradient descent. Adam uses exponentially weighted running averages of the gradients to give the descent "momentum", and it uses exponentially weighted averages of second moments to adapt the learning rates per parameter (so parameters that receive few or small-sized updates learn faster, and parameters with extreme updates have the updates damped).

We used long short-term memory (LSTM) layers, where each "block" combines hidden state vectors from the previous and an input to produce a new hidden state vector. It uses the below "gate"

[1] The dev set was originally intended to be a larger proportion of the dataset, in order to provide more granularity on our validation accuracy for hyperparameter tuning and model selection–but due to a file-naming mix-up we caught after running tuned models on the test set, we were still using a 184-example dev set made for debugging on our local computers. So 184 it was, and we will now both be more organized going forward about syncing between machines.

computations to better remember activations across the sentence.

$$
\begin{aligned}
\text{Input Gates: } i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\
\text{Forget Gates: } f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\
\text{Cells: } c_t &= f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
\text{Output Gates: } o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o) \\
\text{Cell Outputs: } h_t &= o_t \tanh c_t
\end{aligned}
$$

We added dropout layers to lower variance (tuned via hyperparameter search), and used a dense layer with softmax activation at the end to output a classification value from the final hidden state (or the average hidden state, or max, depending on whether the model used pooling of some kind).

4.2 Neuron Activation Analysis

We conducted a neuron activation analysis from one of our RNN models we trained: the 1 layer, 128 neuron LSTM layer with 0.5 dropout ratio. The analysis output a value for each word in the sentence sequentially. The value of each word represents that given words sequence at current point, what would be the possibility it is a 'making-sense' sentence.

The neuron activation is modeled as Figure 1. For each word, an activation value, A_i, is estimated based on its corresponding 128-dimensional LSTM layer output by feeding them into the dense layer and then a sigmoid layer as trained in the RNN model. The output of neuron activation analysis is a number between [0,1], and it represents the neuron activation level for the sentence ends before the current word.

An example of neuron visualization is in figure 2, with the activation level represented in color. The green color represents the input sequence up to that point being predicted to be from the real-phrase dataset, and the red color represents predicting the sequence to be nonsense; white is unsure. This example sentence is a concatenation of a positive labeled sentence ('you always talk back to me') and a negative labeled sentence ('a we your'). From the figure, the color is more green at the first half sentence, turns to red when there's a preposition without an object, back to green when it gets its preposition, and turns to red immediately thereafter as one might expect. This simply example shows that the model has the potential to identify abnormalities when the sentence structure is wrong.

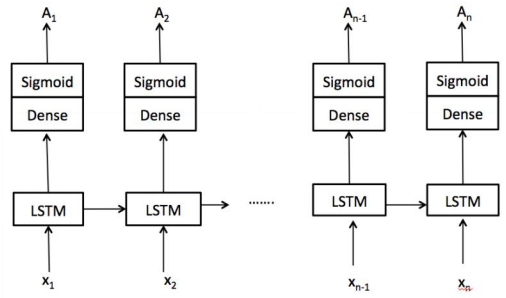

Figure 1: The neuron activation analysis model

you always talk back to me a we your

Figure 2: An example of neuron activation analysis

5 Experiments/Results/Discussion

Our primary metric was accuracy: what percent of the predicted labels matched the ground truth labels. We chose accuracy because the labels were binary, the test set had equal numbers of positive

and negative examples, and recall or precision weren't as germane for our application as they are be in domains like medical test evaluation.

To choice the optimal model, We ran hyperparameter tuning experiments on a small subset of the training data over 20 epochs. We choose small dataset due to the limited computational power we have.

Word model experiments
For the word model experiments we held all hyperparameters fixed but one to find the value of the remaining parameter that maximized accuracy on the dev set, then used that value for that hyperparameter when held fixed for subsequent experiments, with two exceptions: batch size and lstm layers. (We realized that 1 layer model and smaller batches would do better given the relatively small training set used for tuning)

# LSTM layers	state size	dropout	bidirectional	learning rate	batch size	dev accuracy
1	128	0.5	false	0.001	100	0.880
2	128	0.5	false	0.001	100	0.859
2	64	0.5	false	0.001	100	0.837
2	128	0.3	false	0.001	100	0.848
2	128	0.5	true	0.001	100	0.766
2	128	0.5	false	0.001	50	0.897
2	128	0.5	false	0.002	100	0.826

Character model experiments
For the character model model selection and tuning, we implemented global average pooling and global max pooling layers that would work with masking (unlike Keras's provided layers) and included them in the experiments. For the below experiments we used a dropout of 0.2, batch size of 6, and state size of 128.

layers	r. dropout	bidirectional?	pooling	masking	α	dev acc
1	0.0	false	none	masked	0.001	0.652
1	0.0	false	GAP	unmasked	0.001	0.815
1	0.2	false	none	masked	0.001	0.772
2	0.2	false	none	masked	0.001	0.766
1	0.2	false	GAP	masked	0.001	0.782
2	0.2	false	GAP	masked	0.001	0.804
1	0.2	true	none	masked	0.001	0.87
1	0.2	true	none	masked	0.0005	0.831
1	0.2	true	GMP	masked	0.001	0.72

After the hyperparameter tuning experiment, we chose four promising models to train on the entire dataset, and recorded their training accuracy and test accuracy. All four models used 128-dimensional hidden states, 0.5 dropout ratio, 0.001 learning rate and 100 minibatch size. The "fixed-length" model had a fixed RNN length set to just over the max number of words in the dataset inputs, and the others used masking to have just as many as was needed. The results are shown in Table below and discussed more in the conclusions section.

With the best model (figure 3) trained on the entire dataset, we explored the relationship between the size of the dataset and the training and validation accuracy. The results are shown shown in the figure 4 – as the number of data point increases, the training and validation accuracy both improve, indicating that variance is more likely a problem than bias and more data would help further.

Model	Training error (m = 99,080)	Test error (m = 1,663)
Character LSTM	0.3%	8.0%
One-Layer Word LSTM	3.1%	5.54%
Two-Layer Word LSTM	3.2%	5.1%
Two-Layer Fixed-Length Word LSTM	0.65%	3.85%

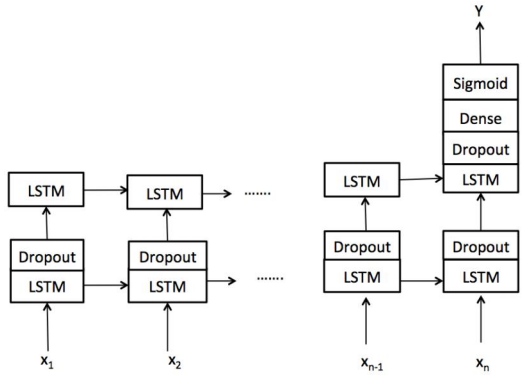

Figure 3: The final model

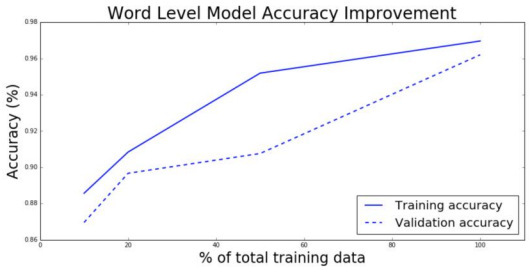

Figure 4: Accuracy and the training data size

6 Conclusion/Future Work

As expected, the word models performed better than the character model, and two layers did better than one. The first was likely thanks to the word embeddings and shorter sequence lengths to "remember" across, and the second to two layers flagging or remembering "nonsensicality" better.

Contrary to our expectations, the fixed-length word LSTM outperformed the others. Its behavior was rather mysterious and worth further analysis. Examining the outputs for inputs of different length, the output would waver as input words were being processed, then dip down to close to 0 ("nonsense") as pad words were read around the middle of the max length, then finally bounce back up to above 0.5 ("real") for positive examples or stay below it for negative examples within 5 cells of the end of the model. It looked like the two layers were approximating a quadratic function of the input number together.

We were unable to find legible correlations between input words and the activation patterns in the hidden states. We found and examined the inputs that caused the greatest activation for each neuron, but no pattern was found. With more time, we might go "[gibber]fishing" for representations by looking for neurons with similar activations across a number of strategically chosen inputs that share a common grammatical structure (or break from structure). We would also look at more of the most-excitatory inputs per neuron, and analyze it in terms of activation for the given word instead of after processing a whole sentence.

Part of the problem was that dropout may have made it less legible, since it would encourage distributing representations across neurons instead of concentrating it in one that could get dropped out.

One extension we would like to do is to do transfer learning on a dataset of slightly perturbed but no-longer-valid input examples, to train a model to help make natural language data augmentation less prone to mislabeling. Unlike in computer vision, data augmentation in natural language processing is a bit risky and complicated, since perturbations that preserve the meaning and syntactic correctness

of one example may break correctness–or worse, change the meaning–when applied to a different example. So being able to filter perturbed inputs would be quite valuable.

We would be careful to avoid confounds as to what the model is learning by having it either try to detect changed meaning or broken syntax, rather than both at once. We might also turn the system into a GAN, or add self-attention to be able to visualize which words were most affecting the output (to see if any subsequences "disqualified" it from being real or "lent it validity" because it was unlikely to appear at random).

7 Contributions

Bo: word level model hyperparameter tuning, weight analysis, activation visualization, literature review lead, plots
Matthew: data pipline, word and character embedding, character level model hyperparameter tuning, and model development
We both worked on the poster and final report.
Abhijeet Shenoi advised us on our project as a course assistant. Matthew's colleague David Liu gave us the idea that our project could be used to make data augmentation more reliable.

References

[1] Fowler, R. On the interpretation of 'nonsense strings', journal of Linguistics 5.1 (1969): 75-83.
[2] Mikolov, T., Karafiat, M., Burget, L., Cernocky, J., Khudanpur, S. (2010). Recurrent neural network based language model. In Eleventh Annual Conference of the International Speech Communication Association.
[3] Zhang, Shu, et al. Bidirectional long short-term memory networks for relation classification." Proceedings of the 29th Pacific Asia Conference on Language Information and Computation. 2015.
[4] Wu, M., Liu, L., Yao, W., & Yin, C. Semantic Relation Classification by Bi-directional LSTM Architecturen.

[5] Pennington, J., Socher, R., & Manning, C.D. (2014) GloVe: Global Vectors for Word Representation.
[6] Hochreiter, S., & Schmidhuber, J. (1997) Long short-term memory. *Neural Computation*, **9** (8): 1735–1780. doi:10.1162/neco.1997.9.8.1735.
[7] Diederik, K & Ba, J (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.
[8] http://downloads.tatoeba.org/exports/sentences.tar.bz2 (CC-BY)